NoSQL Write-up – Dota2 Game Replay Analysis

Team name: PSG.LGD

Yichun Yan
Ziwei Jiang
Yifan Li
Weiqi Wang

October 31, 2019

Contents

1 Datasets

1.1 Background of Datasets

Our team aims at exploring data about a popular and long-lived computer game, Dota2. Dota2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve Corporation. Each game will involve ten players that are divided to two teams play against each other on a same map. Every player chooses one "hero" that has unique abilities and different styles of playing. The aim for each team is to destroy the building called "Ancient" of the other team. Players collect gold to buy powerful items and gain XP by killing players on the other team, destroy building of the other team, or kill creeps.

Figure 1: A screenshot of the game

This game has a large quantity of players, accordingly a huge amount of replays will be produced every day. Also, it's a real-time game, hundreds of operations and some decisions will be made each second, and all of them will have some influence on the final results of the game. Both of the above two features make the game a great subject to big data analysis. Actually, there are some professional teams start hiring data engineers to help their players perform better in a game. This means that our analysis has some realistic meaning. It is a real-time game, hundreds of operations and some decisions will be made each second, and all of them will have some influence on the final results of the game. Therefore, there are some professional teams start hiring data engineers to help their players perform better in a game. For example, by using big data pipeline to answer the business question "Does there exist a regular farming path for some professional players?", we can understand both our teammates and the players from other team better, and therefore optimize our strategies. We have used technologies like MongoDB, Apache Spark, Neo4j for this project. All of the code of this project can be found in this repository, and I will explain the logic of our project and reference to some lines of code when neccesary in the following sections.

1.2 Source of Datasets

Dataset Name	Dataset Source	format
Dota2 match result dataset	Valve's official API	JSON file in this schema
Dota2 replay dataset	Valve's Dota2 replay servers(url to download replays from the servers can be constructed by information obtained by this api and this api.)	`.dem` binary format, can only be executed by Dota2's client, need to be parsed

1.3 Descriptions of Datasets

1. Dota2 match result dataset: Snapshot of some important information about a game and each player, which includes things like which sides win, the duration of the whole game and each player's ending level, items and damage caused.
2. Dota2 replay dataset: Every thing happens in a game, which includes the whole combat log of the game, chatting information and the spawn and death of NPCs.

We will discuss how we acquire those data how it is transformed into a usable format in details in the ETL selection.

1.4 Data Dictionaries

The Dota2 match result data is already in JSON format, we can use it straightforwardly in the same schema to save it into our database, the data dictionary for this dataset should be:

Field Name	Data Type	Description	Example
players	JSON Array	status about each player at the end of the game, I will talk about it in details in the next table.	Will be mentioned in details in the next table.
radiant_win	boolean	Dictates the winner of the match, true for radiant; false for dire.	true
duration	int	The length of the match, in seconds since the match began.	2452
start_time	long	Unix timestamp of when the match began.	1566731204
match_id	long	The matches unique ID.	4986382845
match_seq_num	long	A 'sequence number', representing the order in which matches were recorded.	4182489897
tower_status_radiant and tower_status_dire	int	see this in details https://wiki.teamfortress.com/wiki/WebAPI/GetMatchDetails#Tower_Status	260
barracks_status_radiant and barracks_status_dire	int	see this in details https://wiki.teamfortress.com/wiki/WebAPI/GetMatchDetails#Barracks_Status	51
first_blood_time	int	The time in seconds since the match began when first-blood occurred.	102
lobby_type	int	The type of the match	0
human_players	int	The amount of human players within the match.	10
leagueid	int	The league that this match was a part of. A list of league IDs can be found via the GetLeagueListing method.	0
positive_votes	int	The number of thumbs-up the game has received by users.	0
negative_votes	int	The number of thumbs-down the game has received by users.	1
game_mode	int	The mode of the match	11
picks_bans	JSON Array	A list of the picks and bans in the match, if the game mode is Captains Mode.	I will describe it in details in another table
radiant_score	int	The total amount of kills that radiant team make	52
dire_score	int	The total amount of kills that dire team make	42

The data dictionary for one JSON object of field "players" that mentioned before should be:

Field Name	Data Type	Description	Example
account_id	long	An unique identifier for each steam account	4294967295
player_slot	int	see this in details https://wiki.team-fortress.com/wiki/WebAPI/Get-MatchDetails#Player_Slot	1
hero_id	int	The hero's unique ID. A list of hero IDs can be found via the GetHeroes method.	97
$\text{item}_0 to item_5$	int	item id of the inventory item	48
kills	int	The amount of kills attributed to this player.	10
deaths	int	The amount of times this player died during the match.	2
assists	int	The amount of assists attributed to this player.	7
leaver_status	int	Indicating whether a player disconnect from the game before the game ends	0
gold	int	The amount of gold the player had remaining at the end of the match.	32341
last_hits	int	The amount of last-hits the player got during the match.	109
denies	int	The amount of denies the player got during the match.	24
gold_per_min	int	The player's overall gold/minute.	601
xp_per_min	int	The player's overall experience/minute.	587
gold_spent	int	The amount of gold the player spent during the match.	21239
hero_damage	int	The amount of damage the player dealt to heroes.	34034
tower_damage	int	The amount of damage the player dealt to towers.	8902
hero_healing	int	The amount of health the player had healed on heroes.	452
level	int	The player's level at match end.	23

The data dictionary for one JSON object of field "picks_bans" that mentioned before should be:

Field Name	Data Type	Description	Example
is_pick	boolean	Whether this entry is a pick (true) or a ban (false).	true
hero_id	int	The hero's unique ID.	20
team	int	The team who chose the pick or ban; 0 for Radiant, 1 for Dire.	1
order	int	The order of which the picks and bans were selected.	10

The second dataset is initially a binary file, we have to parse it first then fit into a schema that we design. After

these process, one document(record) of a replay should have the following fields:

1. match_id: A long type field to specify the unique match id of this match, example value can be 4986514901
2. combatlog: A JSON array type field to describe all the combat logs of the game. I will describe what does a JSON object in the array looks like and an example value in another table.
3. info: A JSON object to describe the basic information about the match, I will describe the fields in details in another table.
4. chat: A JSON array type field to show what the players say in the chat during the match, an example of one chat information can be: `{ "sender" : "Sk SpwaN", "message" : "gg wp" }` , where the sender field is the nickname of the player that sends the message, and the message field is just the message the player sends.
5. lifestate: A JSON array type field to indicate the spawns and deaths of objects in the game, an example of one JSON object in this array will be `{"tick" : 7420,"type" : "spawn","object" : "CDOTA_BaseNPC_Creep_Lane"}` . We don't really use this field so I won't explain it in details.

The data dictionary for one JSON object of field "combatlog" that mentioned before should be:

Field Name	Data Type	Description	Example
time	double	The time when this action happens	325.6087341304 32
type	String	The type of this action.	"purchase"
target	String	The target of this action	"npc_dota_hero_nevermore"
item	String	This field only occurs when the type of the action is "purchase". Indicates which item was bought by this action	"item_enchanted_mongo"
inflictor	String	This field only occurs when the type of the action is "damage", "lose_buff" and "add_buff". The inflictor of the current action	"modifier_tower_aura_bonus"
damage	int	This field only occurs when the type of the action is "damage", which indicates the damage caused by the current action.	80
before_hp	int	This field only occurs when the type of the action is "damage", which indicates the hp of the target before the action.	725
after_hp	int	This field only occurs when the type of the action is "damage", which indicates the hp of the target after the action.	718

The more detailed data dictionary of field "info" that mentioned before should be:

Field Name	Data Type	Description	Example
game_winner	int	1 if radiant wins and 2 if dire wins.	1
leagueid	int	The unique league id of the current professional game, 0 if it is not a professional match.	4122
match_id	long	The unique identifier for each game.	4986514901.
end_time	long	The timestamp of the ending of the game.	1566736571.
game_mode	int	The mode of the current game	21

2 Data ETL

Our aim is to collect about 1,500,000 records for our match result dataset and about 1000 professional games, 1000 ranked games and 1000 public games for our replay dataset. Our data is mainly three sources:

1. Valve's official api.
2. OpenDota's third party api.
3. Dota2's replay cluster.

I will walk through how I create a pipeline to make use of the above three sources to collect all the data I want in the Extraction section.

2.1 Extraction

First, in order to call the Valve's official api, we need a key to get authorized. To avoid the api call limits of the api, I create a key pool that contains 5 different authorizing keys that are obtained using our steam accounts. Every time I need to call the api I just randomly pick one key from the poll. Tha api we are mainly using is this one. Which requires two parameters: `start_at_match_seq_num` and `matches_requested` . The first one is an unique identifier for each match, and the api will return a number of results that equals to the second parameter you give to the api in this JSON format. We construct the url for the api call like this to get a bunch of match results and save them into MongoDB. Also, since we always want to keep the HiFi data around, I also wrote the original JSON file into the disk. After we get a list of match result in the previous call, we can use the sequence number of the last record we get from the last call then plus one as the `start_at_match_seq_num` for the next iteration. Also, considering the ratio of professional games is very small, I iterate the whole results to find out whether there is a valid professional game. If I find a valid professional game, I will call OpenDota's API using the `match_id` , which is also an unique identifier of the professional game and can be obtained by the previous results to get the information we need to download the replays. Then we use this information to construct an url like this format `http://replay<cluster>.valve.net/570/<match_id>_<replay_salt>.dem.bz2` and start another thread to to download, unzip and parse the replay. At the same time, since we want to get a same number of ranked matches and public matches, we maintain two integers to record how many public matches and ranked matches to download. As we start downloading a professional match, we add one to those two integers, and if the numbers are not zero when we encounter a public match result or a ranked match, we start to download them using the same method I used for professional games, this logic can be mapped to this snippet of code. This way we can get game replays of three types in same amount for the second dataset while we collect the data for the first dataset.

2.2 Transformation

The match result dataset is easy to handle, since the result of this api is just a JSON string, which can be easily parsed into Document datatype by the MongoDB Driver in Java and saved to MongoDB. But the replay is in binary format, which can only be executed by the Dota2's client. We need to use some open-source parser to help us transform it into something that makes more sense to us. Luckily, we found this parser on GitHub. When use the pipeline that mentioned before to download and unzip each replay and parse it into strings, then use the schema we mentioned in the Data Dictionaries section. The code to parse the strings into our schema is following:

1. The overall logic.
2. For the "info" field.
3. For the "combatlog" field .
4. For the "chat" field.
5. For the "lifestate" field.

2.3 Loading

We deploy our database on AWS EC2, I used this snippet of code to save the JSON format match result data into MongoDB and this snippet of code to write parsed replay data into MongoDB.

3 Data Analysis

In data analysis we used Neo4j to analyze the win-lose relationship between professional games. And use Spark to analyze business questions raised in our proposal. We don't fulfill all business questions and we will state the reason for it in the challenges section. Here are business questions we completed:

1. Bad Manner
   This analysis is aimed to find out the hero which has most possibility to be a bad manner player. We define bad manner as leaving a game before the game ends. We use match result data, since it contains leave game status. We first filter games by their types(public, ranked, or professional). Then we do aggregate on the hero id to get the pairs of a hero id and a leave game count. Before further steps, we first check if there is no bad manner player, which turns out to be true in professional games. Then for each pair, we divide the leave game count by the total pick count of this hero to compute the possibility. Finally, we can get the bad manner hero by sorting the processed pairs. In public game, the most banner hero is Broodmother, who has 10% leave game rate. In ranked game, the most bad manner hero is Meepo, who has 5% leave rate. And in professional games, there are almost no players leave game before a game ends. Through the result, we can find out that players are more serious in ranked games than public, since ranked games affect their rankings.
2. Cost Time
   This analysis is aimed to find out the average cost time for public, ranked, and professional game. We first filter games by their types(public, ranked, or professional). We use the match result data which has the game duration. For each match result data, we can extract the duration and increment the total game count by 1. Finally, we can get the average cost of time by divide the sum of all duration by the total game count. For public games, the average cost of time is 33 minutes. For ranked game, the average time is 40 minutes. For ranked game, the average time is 30 minutes.
3. Damage Rate
   This analysis is aimed to find out the heroes who have top 20 damage rate and the heroes who have bottom 20 damage rate. We define the damage rate of a hero as the damage to other heroes per gold gained by the hero. We use match result data which contains damage to other heroes, gold spent, and rest gold of each

hero. We first filter games by their types(public, ranked, or professional). Then for each match result, we extract the 3tuple of a hero id, a total damage to other heroes, a total gained gold (addition of gold spend and rest gold). Then we do an aggregate on the hero id and divide the damage by the gained gold to get pairs of a hero id and a damage rate. Finally to get the top 20 heroes and bottom 20 heroes, we can simply sort pairs by the damage rate. We list the result for this analysis in Appendix.

4. Economy Distribution vs Game Result
   This analysis is aimed to find out whether there exists correlation between economy distribution in a game and the game result. We define economy as gold gained in a game, since it's limited in a game, we hypothesize that the distribution of gold has effect on the game result. It's possible that an even distribution is better and it's also possible that good players gaining more gold is better. To exam the hypothesis, we apply logistic regression in this analysis. We use the match result data which contains the amount of gold gained per minute for each player. For each match result data, we extract a pair of a game result, a normalized gold distribution. A gold distribution contains the proportion of gold of 5 players in a game. For each distribution, we sort it in ascending order to make the smallest proportion at the head of the distribution. Then we can use sorted distribution as the data point for the logistic regression model. Each proportion in a distribution is a feature. And we use the game result as the label for logistic regression. After running the training data, we find the accuracy is only 0.53 which is close to making a guess. Thus, our hypothesis turns out to be false.
5. Gain in First 15 Minutes
   This analysis is aimed to find out the hero who gains most XP or gold in public, ranked, and professional game. We use replay data which contains all events during a game. We first filter games by their types(public, ranked, or professional) and the then filter by XP or gold. In event, we can extract the time when this event happen. Thus, we can filter event to get only the first 15 minutes events of gaining XP/gold. Then we can do aggregation on hero id to calculate the total gain XP/gold. Finally, we sort the gained XP/gold value to compute the hero who gains most XP/gold in 3 game types. We list the result for this analysis in Appendix.
6. First Blood vs Cost Time
   This analysis is aimed to find out the correlation between the time of first blood and the total cost of time in a game. The first blood is the first kill in a game. We guess a early first blood means a game is fast. To exam our guess, we apply linear regression in this analysis. We use match result data which contains the first blood time and the total duration. For each match result, we extract the first blood time and the game result. The data point used in linear regression is the time of the first blood. The label used is the game result. But the root mean squared error on training data is 765 which is large. The error shows that there is not correlation between the first blood time and the cost of time.
7. Most Stats Hero
   This analysis is aimed to find out the top 5 heroes who has most kills/assists/deaths/heals in public, ranked, and professional game. We use replay data which contains all events during a game. We first filter games by their types(public, ranked, or professional) and then filter by the event type we want (kill/assist/death/heal). Then for each replay data, we extract a pair of a hero id and a count of the event type. Then we do aggregation on the hero id and sort pairs by the count to extract the top 5 heroes. We list the result for this analysis in Appendix.
8. Most Ban
   This analysis is aimed to find out the top 5 heroes who is banned in professional games (only professional game allows banning heroes). We use match result data which contains banned heroes. We first filter games to keep only professional games. Then we extract a pair of a hero id and 1 (denotes banned once) for each match result data. After doing aggregation on the hero id, we sort the banned count to get the top 5 banned heroes. They are Alchemist, Enchantress, Shadow Demon, Faceless Void, and Life Stealer. And we can find some of them has a good ranking in previous analysis. For example, Alchemist has 7th damage rate in public games.

9. Most Pick
   This analysis is aimed to find out the top 5 heroes who is picked in public, ranked, and professional game. We use match result data which contains picked heroes of each game. We first filter games by their types(public, ranked, or professional). Then we extract pairs of a hero id and 1 (denotes picked once). After doing aggregation on the hero id, we sort the picked count to get the top 5 picked heroes. We list the result for this analysis in Appendix.
10. Most Used/Purchased Item
    This analysis is aimed to find out the top 5 most used/purchased items in public, ranked, and professional game. We use replay data in this analysis because it contains all events happen during a game. We first filter games by their types(public, ranked, or professional). Then we filter events contained in replay data to get all 'use item'/'purchase item' events. Then we can reduce them by the item id to get the pairs of an item id and an item use/purchase count. Finally, we can get the top 5 most used/purchased item by sorting the pairs. We list the result for this analysis in Appendix.
11. Team Battle Detector
    This analysis aimed to find out the average first time a team battle and the average times of team battle happens in public, ranked, and professional games. We use the replay data which has all events. But these events doesn't has the location of events. Thus, we roughly define the team battle as in 3 minutes there are 4 or more hero deaths. We apply sliding-window algorithm to build a team battle detector. We first filter games by their types(public, ranked, or professional). Then we extract the death events and use the detector to detect all team battles in a game. The time of the first team battle is at 11 minutes, 13 minutes and 19 minutes in public, ranked, and professional game. The average number of team battles is 7, 9, and 6 in public, ranked, and professional game.

4 Challenges

I will walk through the challenges we met during this project in the following subsections.

4.1 Valve API

Valve provides us with various apis to get match results, but they're not well documented and changes all the time without documentation. At the beginning I tried to use this api, which is very straightforward. This api requires us to specify a starting match ID and collect a bunch of match results starting with that match ID. But unfortunately I couldn't use it now since Valve didn't allow us to specify such an ID and it can just return the newest results that are currently produced, and it was never documented. I have thought that I could use this api to make a data stream for stream analysis, but this couldn't be done to since Valve limited its api calling gap. It's said to be 100,000 calls per day but actually you can do much less. I set the api calling interval to 6 seconds and sometimes it still denies the calls. Considering the velocity that the game results are produced we can't make such a stream to catch all the game results that are being produced. After exploring the Dota2 developers' forum for a while, I found out that we can use this api, which works similarly as the before one but this time we need to specify a starting sequence number, which is also a unique identifier for each game but we can't see it in the Dota2's client so it gives us more difficulty. Later I found out that I can find the sequence number in this website. So I can first choose a starting sequence number by looking for a sequence number of a match whose date is around the starting date of all the matches I want to collect. By calling this api, we can get a bunch of match results, so we can extract the sequence number of the last match of the results and plus 1 to it as the starting sequence number of the next request to the api. By doing this repeatedly, we can collect as many matches we want. As I mentioned, the api call limits are much less than it is described and it seems to be random. I created a request key pool to randomly select a key for the api call each time and if I was denied, I waited 3 seconds then do the next call.

4.2 Dota2 replays

The replay dataset can be constructed by using url like this format: `http://replay<cluster>.valve.net/570/<matchid><replaysalt>.dem.bz2` . In the past, by calling this api using match ID of a game to get the cluster number and "replay salt" for a specific match. But now Valve disallowed us to do this too by hiding the "replay salt" field it the result due to the pressure on it's replay clusters is too large. Fortunately, I found another https://docs.opendota.com/ that can help me find the fields I need to download a replay. So I thought of purely rely on this api to get the data in the first dataset, this way I can only use one kind of api to simplify the data pipeline. But this can't be done too, this third party api will ask us to pay if our request exceeds a certain number, that will cause cost exceeding our budget(which is zero). Also, we don't want to collect all the replays. It's clear that the matches played by the professional teams worth more than the normal ones. Given the limited time and storage we have, I think I will first collect a certain number of professional games, then collect public games and ranked games of the same number for comparing analysis with the professional games.

4.3 Docker

As mentioned in the proposal, we tried to containerized the project by docker, but it ends up with failures. Considering to the limited time, we decided to discard using docker for installing our NoSQL technology.

At first, I tried to dockerize our project by using a single Dockerfile. According to the docker documents, Dockerfile "defines what is going on in the environment inside your container". At the very beginning, I thought dockerize a project just need to pull all the official images in docker, and then we can start manipulate it. But then, after reading documents, I found that we need to construct a Dockerfile to define how a project work, so that it can behaves exactly the same wherever it runs. We can pull an image as a parent image, and then install needed packages in it.

For example, in the project, I used ubuntu as a parent image, then docker virtualized an ubuntu operation system environment on our host machine, and then I used Dockerfile as if it is the terminal of ubuntu, writing codes of installing java8, maven, mongodb in the Dockerfile.

After I constructed the Dockerfile, I failed every time when it was trying to connect to the database. Realized that it is cramped to put all the services on a Dockerfile, I tried to construct a docker-compose.yml to define, run and scale services with Docker.

According to our data pipeline, I intended to build 4 services in the docker-compose:

- Data acquisition and filtering (as "fetchstore" service in the docker-compose)
- Data Extraction (as "parsereplay" service in the docker-compose)
- Data Aggregation and Representation (as "mongodb" service in the docker-compose)
- Data Analysis (as "analytics" service in the docker-compose)

In the docker-compose, we can also use the Dockerfile by using the command `build` in a service. Therefore, I constructed Dockerfiles for Extraction, Transformation and Analysis services. And I used the official image of mongodb, and used `volume` in the service to reflect the direction of configuration file and mongodb initial javascipt. Also, we used `depends_on` to define simple interaction between the services.

As the result, we still have a hard time connecting the database. There are a lot of things to be considered when running multiple containers. When this problem is fixed, we only finished the configuration of the extraction and database service. Then we realized we got limited time left, therefore, we discard docker for now, and move on to deploy our project on amazon web service (AWS). And we hope to complete this part in the future.

The reason of failure is that I got misunderstanding on Docker's architecture. Also, it is hard for me to consider

our system's architecture in Docker's way. Also it is time consuming to try those failures to realize the right way to deploy a project on Docker. This experience pushed me to learn a lot about Linux and to read a lot of configuration file.

Glossary

Ancient (also commonly refered to as Thrones, or Tree for Radiant's ancient and Throne for Dire's ancient, as legacy names from DotA) are massive structures found inside each faction's base and are the main objective. In order to win, the enemy team's Ancient must be destroyed, while the own one must be kept alive. Ancients are guarded by their two tier 4 towers. The Ancients are invulnerable until both of their tier 4 towers are destroyed.. 2

creeps Creeps are basic units in Dota 2. Every unit which is not a hero, building, ward or courier is considered a creep. Creeps can belong to either faction, be neutral, or be player-controlled units. Unlike heroes, creeps do not gain experience and cannot level up. All of their stats are set values (though can still be altered by modifiers). Most creeps grant a set gold and experience bounty to heroes when killed.. 2

gold Gold is the currency used to buy items or instantly revive your hero. Gold can be earned from killing heroes, creeps, or buildings.. 2

MOBA also known as action real-time strategy (ARTS), is a subgenre of strategy video games that originated as a subgenre of real-time strategy in which each player controls a single character, usually on a map in an isometric perspective, as part of a team competing against another team of players.. 2

XP A shorthand for experience. Experience is an element heroes can gather by killing enemy units, or being present as enemy units get killed. On its own, experience does nothing, but when accumulated, it increases the hero's level, so that they grow more powerful. Only heroes can gather experience and therefore reach higher levels. With each level gained, a hero's base attributes increase by static values (unique for each hero), which makes them stronger in several.. 2

5 Appendix

1. damage rate
2. Gain in First 15 Minutes
3. Most Stats Hero
4. Most Pick
5. Most Purchased Item
6. Most Used Item